METHOD AND APPARATUS FOR PROVIDING DYNAMIC SECURITY MANAGEMENT

Field of the invention

5   The present invention relates to a method for providing dynamic security management in an apparatus, especially a mobile telephone terminal. The invention enables download of applications and shared libraries with associated enhanced security policy. By means of the invention, the security policy is also enhanced by a more detailed structure enabling access of functions at various levels. The invention
10   also relates to an apparatus implementing the method.

State of the art

         Today mobile telephone terminals include functions implemented in software and hardware that may be accessed by applications implemented in e.g. a Java
15   platform through an application programmer's interface (API). The access is controlled by a security manager using access permissions stored in a security policy. The access permissions are defined for specific domains (also called sandboxes in a sandbox model). The applications are linked to a specific domain through a root certificate. The applications are integrity protected and their origins are
20   authenticated using digital signatures and certificates pointing to the root certificate.

         Thus, functions required for the applications are linked to specific functions existing in the terminal by means of the access permissions as defined by a domain to which the application is linked. However, the security policy is static and it is only possible to download new applications as long as the security policy includes
25   the required access permissions for new functionality associated with the new applications.

         It may be desired to download functionality where the security policy in the specific terminal does not include the required access permissions. For instance, the terminal functions may be completed and built into the telephone before the Java
30   platform and security policy is finished. Another problem is that the static security policy does not allow components shared between different applications. The present invention resolves the problem by modifying the security model to enable downloading of applications and shared components, e.g. libraries, together with associated policy information. Even policy information as such may be
35   downloaded.

Summary of invention

         In a first aspect the invention provides a method of providing a dynamic security management in an apparatus comprising: a platform for running an

application; a security manager for handling access of the application to functions existing in the apparatus; an application interface (API) between the platform and the application; a set of access permissions stored in the apparatus and used by the security manager for controlling access of the application to functions through the

5   application interface.

According to the invention, the method comprises the steps of:
downloading into the apparatus an object containing access permissions applicable to at least one function;
verifying the object;

10   installing the access permissions together with the existing permissions.

Suitably, the object is verified by checking a certificate chain of the object and it is verified that a policy of the function allows updates.

In one embodiment, the method comprises downloading a further object containing a library, or the downloaded object further containing a library, said

15   library comprising new routines and/or new functions to be called by an application or library stored in the apparatus; and installing the library to enable access of functions through the application interface (API).

Preferably, the new routines and/or new functions can access existing functions through a library.

20   The security manger, when accessing functions, may recursively check the permissions of the APIs and libraries in a linked chain related to the called functions

In a further embodiment, the method comprises downloading a further object containing an application, or the downloaded object further containing an application, said application containing at least one new function; and installing the

25   new function so that the new function can access existing functions through the application interface.

Preferably, the new functions can access existing functions through a library.

Suitably, the access permissions are contained in a policy file.

Preferably, the policy file has a structure linking access levels of existing

30   functions with a domain associated with the downloaded object.

Preferably, the policy file also has a structure linking access levels of existing functions with information contained in a certificate chain.

The information may include signature of the end entity certificate, signature of an intermediate certificate, or specific level information (level OID).

35   The policy file may have a structure including logical expressions.

In a second aspect the invention provides a method of providing a dynamic security management in an apparatus comprising: a platform for running an application; a security manager for handling access of the application to functions existing in the apparatus; an application interface between the platform and the

application; a set of access permissions stored in the apparatus and used by the security manager for controlling access of the application to functions through the application interface.

According to the invention, the method comprises the steps of:

5   storing the access permissions in a security policy;
providing the security policy with a hierarchical structure.

Preferably, the security policy has a structure linking access levels of existing functions with a domain associated with the downloaded object

Preferably, the security policy also has a structure linking access levels of

10 existing functions with information contained in a certificate chain.

The information may include signature of the end entity certificate, signature of an intermediate certificate, or specific level information (level OID).

In a third aspect the invention provides an apparatus with dynamic security management comprising: a platform for running an application; a security manager

15 for handling access of the application to functions existing in the apparatus; an application interface between the platform and the application; a set of access permissions stored in the apparatus and used by the security manager for controlling access of the application to functions through the application interface.

According to the invention, the apparatus is arranged:

20 to download an object containing access permissions applicable to at least one function;
to verify the object; and
to install the access permissions together with the existing permissions.

Suitably, the security manager is adapted to verify the object by checking a

25 certificate chain of the object, and to verify that a policy of the function allows updates.

In one embodiment, the apparatus is arranged to download a further object containing a library, or the downloaded object further containing a library, said library comprising new routines and/or new functions to be called by an application

30 or library stored in the apparatus; and to install the library to enable access of functions through the application interface.

Preferably, the new routines and/or new functions can access existing functions through a library.

Suitably, the security manger, when accessing functions, is adapted to

35 recursively check the permissions of the application interfaces and libraries in a linked chain related to the called functions.

In a further embodiment, the apparatus is arranged to download a further object containing an application, or the downloaded object further containing an application, said application containing at least one new function; and to install the

4

new function so that the new function can access existing functions through the application interface.

Preferably, the new functions can access existing functions through a library.

Suitably, the access permissions are contained in a policy file.

Preferably, the policy file has a structure linking access levels of existing functions with a domain associated with the downloaded object.

Preferably, the policy file also has a structure linking access levels of existing functions with information contained in a certificate chain.

The information may include signature of the end entity certificate, signature of an intermediate certificate, or specific level information (level OID).

The policy file may have a structure including logical expressions.

In a third aspect the invention provides an apparatus of providing a dynamic security management in an apparatus comprising: a platform for running an application; a security manager for handling access of the application to functions existing in the apparatus; an application interface (API) between the platform and the application; a set of access permissions stored in the apparatus and used by the security manager for controlling access of the application to functions through the application interface (API).

According to the invention, the apparatus is arranged to:

store the access permissions in a security policy;

provide the security policy with a hierarchical structure.

Preferably, the security policy has a structure linking access levels of existing functions with a domain associated with the downloaded object.

Preferably, the security policy also has a structure linking access levels of existing functions with information contained in a certificate chain.

The information may include signature of the end entity certificate, signature of an intermediate certificate, or specific level information (level OID).

The apparatus may be a portable telephone, a pager, a communicator, a smart phone, or an electronic organiser.

Brief description of the drawings

The invention will be described in detail below with reference to the accompanying drawings in which

Fig. 1 is a schematic drawing illustrating a security framework based on the sandbox model;

Fig. 2 is a schematic drawing illustrating the operation of the security manager;

Fig. 3 is a schematic drawing illustrating the process of downloading a library object;

Fig. 4 is a schematic drawing illustrating the process of downloading an application object;

Fig. 5 is a schematic drawing illustrating the process of downloading a policy file object;

5   Fig. 6A is a schematic drawing illustrating the structure of the security policy;

Fig. 6B is a schematic drawing illustrating an example of a security policy;

Fig. 7 is a flow chart of an access procedure.

10 Detailed description of preferred embodiments

The invention is described with reference to downloading of applications in an mobile telecommunication apparatus such as a telephone. The apparatus may be a portable telephone, a pager, a communicator, a smart phone, or an electronic organiser.

15   Within the wireless industry MIDP (Mobile Information Device Profile) 2.0 is more or less a de facto standard. The security framework in MIDP 2.0 is based on signed applications and a sandbox model as is illustrated in fig. 1. In the terminal or phone 1, an application (Midlet) 2 has been downloaded via the Internet 4 from a MIDlet developer 5 and is located within a sandbox 6. In MIDP 2.0 several virtual

20 sandboxes, domains, can be defined. Each domain defines access to a set of native functions 3. Some of these functions may be sensitive and access is therefore protected. These terminal built-in functions are implemented in hardware and software, usually programmed in the language C, but any implementation is possible within this invention, such as Java code. The applications 2, MIDlets, are

25 integrity protected and their origin is authenticated using digital signatures and certificates. In the prior art a MIDlet was assigned to exactly one domain based on the root certificate that was used to verify its certificate chain. In other words, each domain was associated/defined by a set of root certificates.

The native functions 3 are protected at the edge of the sandbox 6 by a

30 security manager 7 as is shown in fig. 2. The MIDlets are to be run in a platform or virtual machine. The security manger 7 controls the virtual machine's (VM) access of terminal functions 3 through an API 11 (application programmer's interface). If the platform is implemented in Java, the API is a Java API. The security manger 7 checks the permissions of a MIDlet against a security policy 8 predefined in the

35 phone and controls the API 11 accordingly. In the prior art this security policy was static. In the present invention the API (11A) is enhanced by access permissions and security policy files downloaded together with applications MIDlet 1 to MIDlet n (2) and with Library to Library m (12), in fig. 2 represented by the boxes API 1 to API n (11B). There can be several layers of libraries. In the present invention, the

security manager recursively works down the layers. The security manger 7 first checks the permissions of the first API. If this API is calling further APIs and libraries further permissions belonging to those APIs and libraries are checked.

As mentioned in the introduction it is desired to have the possibility to add
5   further functions to the phone by downloading applications and libraries and shared components.
One example of when libraries are useful is for games. Basically, one would like the game-engine to be a library, which could be used by the levels, where the levels could be downloaded as independent applications. Another possibility would be that
10 all game levels are applications and the common library part is included in the level 1 application.

Another example is access to native functionality. In the prior art it was not possible to add a library that gave the Java environment access through a Java API to for example the calendar data base.
15 Even if there was communication between MIDlets a static security model would not fit with this more dynamic environment.

A typical object to be downloaded is an application or MIDlet. It contains code for new functionality, usually written in Java, and contains a user interface with which the user can interact with the new functions. It usually also contains the
20 required policy files that are required for accessing the terminal functions used by the new functionality. The object may be downloaded via the Internet by a user or as a step in the manufacturing process by the manufacturer or another entity customizing the mobile terminal before distribution. Some applications may be shared with other applications already stored in the terminal, thus serving as a
25 library. When the application is installed it enhances the existing API with new functionality.

Another object to be downloaded is a library. This is similar to an application but does not include any user interface. Otherwise it contains new functionality shared with other applications in the same way as a MIDlet.
30 It is also possible to download policy files as such. This is essential when adapting the terminal to functions already downloaded or to be downloaded in the future.

As is known in the art, objects may be integrity protected and their origin may be authenticated using digital signatures and certificates. Each object may
35 contain one certificate or a certificate chain ending with an end entity certificate and finally pointing to a root certificate stored in the terminal, or on a separate module like a SIM-card. Each certificate contains a signature of an organization which can be linked to the domain through the root certificate. Also additional information may be added, each object being identified e.g. with an OID (Object Identifier). The
40 present invention suggests adding a level OID to be used as described below. A

7

downloaded object is contained in two file objects, a Jad (Java Application Descriptor) containing the certificate chain and signatures and a Jar (Java Archive) containing the files proper such as the policy files containing access permissions and class files containing program code.

5   The structure of a library including a Jad file 9A and a Jar file 10A is shown in fig. 3. The MIDlet permissions contain the domain definition and other permission information to be associated with the policy contained in the downloaded policy files as well as policy files already existing in the terminal. When a library is downloaded and installed the following steps are performed:

10   1.   Verify the certificate chain in the Jad 9A
     2.   Verify the signature in the Jad 9A against the Jar 10A
     3.   Give the library permissions to use functions and other libraries based on the root certificate etc. (must include all parameters we define)
     4.   Read the policy file that should apply for this library and install it together
15   with the library

The structure of a MIDlet application including a Jad file 9B and a Jar file 10B is shown in fig. 4. The MIDlet permissions contain the domain definition and other permission information to be associated with the policy contained in policy files already existing in the terminal. MIDlet applications may also have the same
20   permission file (policy files) included in the Jar-file, i.e. the application can then be used both as an application and a library. The downloaded MIDlet application shown in figure 4 without policy files requires that permissions have been stored previously in the terminal, for instance by downloading a policy file as is discussed below with reference to figure 5.

25   When a MIDlet is downloaded and installed the following steps are performed:
     1.   Verify the certificate chain in the Jad 9B
     2.   Verify the signature in the Jad 9B against the Jar 10B
     3.   Give the MIDlet permissions to use functions and libraries based on the root
30   certificate etc. (must include all parameters we define)

The structure of an object only containing policy files is shown in fig. 5. The object includes a Jad file 9C and a Jar file 10C. When a policy file is downloaded and installed the following steps are performed:
     1. Verify the certificate chain in the Jad 9C
35   2. Verify the signature in the Jad 9C against the Jar 10C
     3. Find the libraries or native functions to which this policy shall apply
     4. Verify that the policy of the libraries or native functions allows policy updates from the current source

The policy for allowing policy updates uses the same syntax as the one

8

describing API access permissions (as is described below with reference to figure 6A). Basically the domain, CA, level or end entities can be used to determine if policy updates are allowed from the current source. In order to associate the policy with a library, the policy file must contain the identity of the libraries it is to be
5   associated with.

Several methods can be used to identify a certificate, the subject field, hash of the public key etc. The best method is to use issuer and serial number which is unique for each certificate.

The downloaded objects form the new APIs 1 to n, denoted by 11B in figure
10   2, and the libraries1 to m, denoted by 12 in figure 2. This enhanced application interface gives rise to an "API" with several layers in which functions may be linked through various APIs and libraries. The security manger 7 recursively checks the permissions of the APIs and libraries in a linked chain related to the called functions.

15   The security policy contains access permissions for various functions, e.g. "Function_A" in fig. 6A. The access permissions are associated with access levels indicating to which extent the user is involved when accessing a function. There are conventionally four levels: "allowed", that is the user has full access and is never asked; "blanket", that is the user is asked once during a period in which the
20   application is installed; "session", that is the user is asked only once each time the application is executed; and "one shot", that is the user is asked at each access.

The security policy may have a structure as shown in fig. 6A. As is shown the security policy has a hierarchical structure. In the prior art the access right of function was only linked with a domain as identified by the root certificate. In the
25   present invention the security policy may be enhanced by more detailed information. The domain defines the basic access level which may be combined with other information to define a more fine grained access level possibly resulting in a higher or lower access level of the function than defined by the domain. For instance, a level OID may be defined in the end entity certificate, one certificate
30   may be signed by a certificate authority (CA) and the end entity certificate may be signed by specific organization defining a specific access level.

An example of a security policy of a Function denoted as "javax.microedition.sonyericsson.CoolNativefunction" is shown in fig. 6B. The subject fields have been assigned specific values, e.g. the subject field of "ca" is
35   Sony Ericsson R&D CA. In plain text, the security policy of fig. 6B states the following:

–   All applications in the manufacturer domain will have at least one shot-access to Function.

–   Applications in the manufacturer domain that have a level 1 indication in the

9

end entity certificate (in other words, an OID value of 1 in the trusted usage
attribute field) will have session access to Function.

–   Applications signed with the certificate issued by Sony Ericsson R&D CA
will have at least session access to Function.

5   –   Applications in the manufacturer domain signed with a key associated with a
Sony Ericsson core Java Team Certificate will have full access (allow) to
Function.

The figure is just an illustration of the hierarchical structure and the exact
representation of the policy file could be in any format, such as XML, attribute

10   certificates etc. Furthermore, the policy can be expressed using logical expressions,
e.g. including AND and OR in the policy file.

An example of a logical expression is e.g. ((InManufacturerDomain AND
(EndEntityCertificate.OID("theOID)="theValue")) OR
((EndEntityCertificate.OID(Organization)="aSpecificCompany") AND

15   (EndEntityCertificate.OID(OrganizationalUnit)="theDepartmentName"))

A possible enhancement to policy file syntax is to allow wildcards when
representing the subject field of the certificate, e.g. Sony Ericsson* would apply to
our R&D certificate, Core Java Team certificate etc.

The level field in the policy file could be matched against an OID in the

20   trusted usage attribute field of the certificate. The level assigned to the application
will be determined by searching the chain for a certificate with a level OID from the
end certificate towards the root. The first one found will be used. Consistency is
checked from the root towards the end entity certificate.

A policy file can contain several policies. Thus, the access level of a specific

25   function may be defined differently in various domains.

The procedure for accessing a function or library is described below with
reference to fig. 7.

In step 1 a MIDlet is calling a function (a native function or any other
function existing in the phone accessible through the API 11 or a library 12 (API 1

30   to API n), as the case may be).

In step 2 the security manager retrieves permission information of the
MIDlet (corresponds to the downloaded MIDlet permissions) by noting the domain
of the associated root certificate, noting any level OID-value, CA signatures and
end entity signatures etc.

35   In step 3 the security manager searches the policy files for access
permissions. In other words, the security manager pairs retrieved permission
information with the access information pertaining to the function of the security
policy.

If any access permission of the function is not found in step 4 access is

10

denied in step 5.

If the function is found, the access level of the function permission is determined. This can be either directly (one hit only) or by resolving a logical expression in the security policy, as in step 7 or, alternatively, a security manager
5   compares access levels of various function permissions pertaining to the specific function and selects the appropriate access level in step 8.

Finally, in step 9 the security manager recursively controls the APIs or libraries using the determined access level of the function. Thus, if the wanted function requires access through further APIs or libraries the procedure is repeated.
10   The user is involved in accordance with the determined specific access level.

The signature scheme we are using gives us integrity protection and origin authentication but not replay protection. To achieve replay protection the signature must be calculated over some fresh data. This can be accomplished by including a timestamp in the Jar file. Note that this method applies also to download of
15   applications and libraries.

For security reasons it is useful if the validity of a policy file can be defined. This would limit the lifespan of rouge policy files. Here we introduce some new elements in the policy file syntax as shown by the validity information at the bottom of figure 6A.
20   Policy file validity is one measure against rouge policy files. A better although more costly method is to do the equivalent of revocation checking when the policy file is installed or used. At install time the policy revocation information can be carried in the Jar file. When the policy is used revocation must be checked against information stored in the device or against an on-line server.
25   Some modifications to our scheme are needed to make it fulfil operator requirements. One requirement is that the policy file should reside on the SIM. The normal case would be to include a URL in the library Jar file pointing at a location on the SIM. One could in addition allow pushed (WAP push, SMS, SAT etc) policy
.   Jad files pointing at a location on the SIM. In an alternative, the same location on
30   the SIM is always used. When performing an update, this position is referenced and just the data is updated, i.e. the policy file. With these methods the operators can remotely modify the policy file on the SIM using conventional methods, such as the SIM tool kit, and then trigger the phone to use it by sending a push message to the phone.
35   Another requirement is to only enable policies from the operator that has the active SIM in the phone. Similarly to the MIDP 2.0 method for enabling and disabling applications we could store the hash of the operator root from the SIM together with the policy file on the phone. Each time a new SIM is inserted the hash of the root would be calculated and checked against the value stored together with

11

the policy files. If they don't match the policy file is disabled.

Thus, the invention provides several advantages over the prior art. The scope of the invention is only limited by the claims below.